# 實驗五 7-Seg LED

0410137 劉家麟 0416324 胡安鳳

## 1. 實驗目的

- 了解 MAX7219 使用原理
- 設計 7-Seg LED 程式

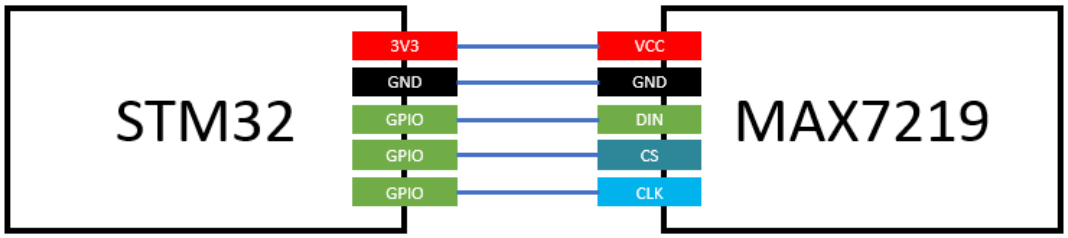
## 1. 實驗原理

請參考上課 lab5_note 講義。

## 2. 實驗步驟

### 2.1. Lab5.1: Max7219 與 7-Seg LED 練習—without code B decode mode

將 stm32 的 3.3V 接到 7-Seg LED 板的 VCC，GND 接到 GND，並選擇三個 GPIO 接腳分別接到 DIN、CS 和 CLK。




完成以下程式碼，並利用 GPIO 控制 Max7219 並在 7-Seg LED 上顯的第一位依序顯示 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, b, C, d, E, F (時間間隔 1 秒)，範例影片如下：

https://goo.gl/ZDZcdl

Note:由於 decode mode 無法顯示 AbCdF 等字，因此請將 decode mode 關掉。(參考 lab5_note 講義的 table 6)

Connect 3.3V and GND pin on STM32 to VCC and GND port on MAX7219. Choose three GIPO ports on STM32 for DIN, CS and CLK on MAX7219.

Complete the code giving below and display 0, 1, 2, 3…, 9, A, b, C, d, E,

F to the first digit of 7-Seg LED at 1 second interval. Example video link is giving above.

Note: Due to the fact that decode mode is unable to display alphabets, please disable decode mode(ref: lab5_note table 6).

```
   .syntax unified
   .cpu cortex-m4
   .thumb
.data
   arr: .byte 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
0x0, 0x0, 0x0, 0x0, 0x0, 0x0 //TODO: put 0 to F 7-Seg LED pattern
here

.text
   .global main

main:
    BL   GPIO_init
    BL   max7219_init
loop:
    BL   Display0toF
    B loop

GPIO_init:
   //TODO: Initialize three GPIO pins as output for max7219 DIN, CS
and CLK
   BX LR

Display0toF:
   //TODO: Display 0 to F at first digit on 7-SEG LED. Display one
per second.
   BX LR

MAX7219Send:
    //input parameter: r0 is ADDRESS , r1 is DATA
   //TODO: Use this function to send a message to max7219
   BX LR

max7219_init:
   //TODO: Initialize max7219 registers
   BX LR

Delay:
   //TODO: Write a delay 1sec function
   BX LR
```

## 1. GPIO_init

Set GPIO here, just like LAB4.

```
RCC_AHB2ENR -> enable GPIOA
 = 0x1

GPIOA_MODER -> set PA5, 6, 7 as output mode
  =(GPIOA_MODER & 0xFFFF03FF) | 0b0101010000000000

GPIOA_OSPEEDER -> set PA5, 6, 7 as high speed
```

```
 =(GPIOA_OSPEEDER & 0xFFFF03FF) |
0b10101000000000000
```

## 2. MAX7219_init

Set decode mode (whether decode or not), intensity (brightness of 7-segment), scan limit (how many digit to display), shut down, and display test.

```
DECODE(0xX9) -> No Decode(0x0)
 = 0x1900

DISPLAY_TEST(0xXF) -> normal operation(0x0)
 = 0x1F00

INTENSITY(0xXA) -> 21/32(0xA)
 = 0x1A0A

SCAN_LIMIT(0xXB) -> only light up digit 0(0x0)
 = 0x1B00

SHUT_DOWN(0xXC) -> normal operation(0x1)
 = 0x1C01
```

## 3. MAX7219Send

We use r0 as higher 8 bits (D15-D8), and r1 as lower 8 bits (D7-D0). Because we use some registers to represent DIN, CS, CLK, BSRR, BRR and the index, we push r0-r7 and LR to stack first and pop them up at last preventing registers from being overwritten.

```
MAX7219Send:
   //input parameter: r0 is ADDRESS , r1 is DATA
   //TODO: Use this function to send a message to
max7219
   push {r0, r1, r2, r3, r4, r5, r6, r7, LR}
   lslr0, 8 //move to D15-D8
   add r0, r1 //r0 == din
   ldr r1, =DIN
   ldr r2, =CS
   ldr r3, =CLK
   ldr r4, =GPIOA_BSRR //-> 1
   ldr r5, =GPIOA_BRR //-> 0
   ldr r6, =0xF //now sending (r6)-th bit
```

```
send_loop:
    mov r7, 1
    lsl r7, r6
    str r3, [r5] //CLK -> 0
    tst r0, r7 //same as ANDS but discard the result
(just update condition flags)
    beq bit_not_set //the sending bit(r0) != 1
    str r1, [r4] //din -> 1
    b if_done

bit_not_set: //send clear bit
    str r1, [r5] //din -> 0

if_done:
    str r3, [r4] //CLK -> 1
    subs r6, 0x1
    bge send_loop
    str r2, [r5] //CS -> 0
    str r2, [r4] //CS -> 1
    pop {r0, r1, r2, r3, r4, r5, r6, r7, PC}
    BX LR
```

4. **Delay**

Same as LAB 4. Initial r0 as one_sec and keep subtracting it until r0 = 0.

```
Delay:
    //TODO: Write a delay 1sec function
    beq  delay_end
    subs r0, 0x4
    b    Delay

delay_end:
    bx   lr
```

5. **Display0toF**

Load the array we'd like to display, and use r0 to save the array index. Because we only display digit 0, the r0 which saving higher 8 bits is 1. And the lower 8 bits load from array. Do the loop and each delay 1 sec to display from 0 to F.

```
Display0toF:
    //TODO: Display 0 to F at first digit on 7-SEG
LED Display one per second
    mov r2, 0x0
    ldr r3, =arr
display_loop:
    mov r0, 0x1
    ldrb r1, [r3,r2]
    bl MAX7219Send

    ldr r0, =one_sec
    bl Delay

    add r2, 1
    cmp r2, 0x10
    bne display_loop
    b  Display0toF
```

6. **Demo Video**
   https://youtu.be/Nk4KFui3HAE

### 2.2. Lab5.2: Max7219 與 7-Seg LED 練習—use code B decode mode

利用 GPIO 控制 Max7219 並在 7-Seg LED 上顯示自己的學號，例如學號為 1234567 則顯示下圖：

完成以下程式碼，將放在 student_id array 裡的學號顯示到 7-seg LED 上。

Note: 請使用 decode mode

Using GPIO output to display your student ID on 7-Seg LED. Picture

above is showing the case that your student ID is 1234567.

Complete the code giving below. Put your student ID in **student_id array** and display it to 7-Seg LED.

Note: Please enable decode mode.

```
   .syntax unified
   .cpu cortex-m4
   .thumb

.data
   student_id: .byte 1, 2, 3,4, 5, 6, 7 //TODO: put your student id
here

.text
   .global main

main:
    BL   GPIO_init
    BL   max7219_init
    //TODO: display your student id on 7-Seg LED
Program_end:
   B Program_end

GPIO_init:
   //TODO: Initialize three GPIO pins as output for max7219 DIN, CS
and CLK
   BX LR

MAX7219Send:
   //input parameter: r0 is ADDRESS , r1 is DATA
   //TODO: Use this function to send a message to max7219
   BX LR

max7219_init:
   //TODO: Initial max7219 registers.
     BX LR
```

1. **GPIO_init**

   Same as LAB 5.1.

```
RCC_AHB2ENR -> enable GPIOA
 = 0x1

GPIOA_MODER -> set PA5, 6, 7 as output mode
  =(GPIOA_MODER & 0xFFFF03FF) | 0b0101010000000000

GPIOA_OSPEEDER -> set PA5, 6, 7 as high speed
  =(GPIOA_OSPEEDER & 0xFFFF03FF) |
 0b0101010000000000
```

2. **MAX7219_init**

   Set decode mode (whether decode or not), intensity (brightness of 7-

segment), scan limit (how many digit to display), shut down, and display test.

```
DECODE(0xX9) -> Code B decode for digit 0-7(0xFF)
 = 0x19FF

DISPLAY_TEST(0xXF) -> normal operation(0x0)
 = 0x1F00

INTENSITY(0xXA) -> 21/32(0xA)
 = 0x1A0A

SCAN_LIMIT(0xXB) -> only light up digit 0(0x0)
 = 0x1B00

SHUT_DOWN(0xXC) -> normal operation(0x1)
 = 0x1C01
```

## 3. MAX7219Send

Same as LAB 5.1.
We use r0 as higher 8 bits (D15-D8), and r1 as lower 8 bits (D7-D0). Because we use some registers to represent DIN, CS, CLK, BSRR, BRR and the index, we push r0-r7 and LR to stack first and pop them up at last preventing registers from being overwritten.

```
MAX7219Send:
   //input parameter: r0 is ADDRESS , r1 is DATA
   //TODO: Use this function to send a message to
max7219
   push {r0, r1, r2, r3, r4, r5, r6, r7, LR}
   lslr0, 8 //move to D15-D8
   add r0, r1 //r0 == din
   ldr r1, =DIN
   ldr r2, =CS
   ldr r3, =CLK
   ldr r4, =GPIOA_BSRR //-> 1
   ldr r5, =GPIOA_BRR //-> 0
   ldr r6, =0xF //now sending (r6)-th bit

send_loop:
   mov r7, 1
   lsl r7, r6
   str r3, [r5] //CLK -> 0
   tst r0, r7 //same as ANDS but discard the result
```

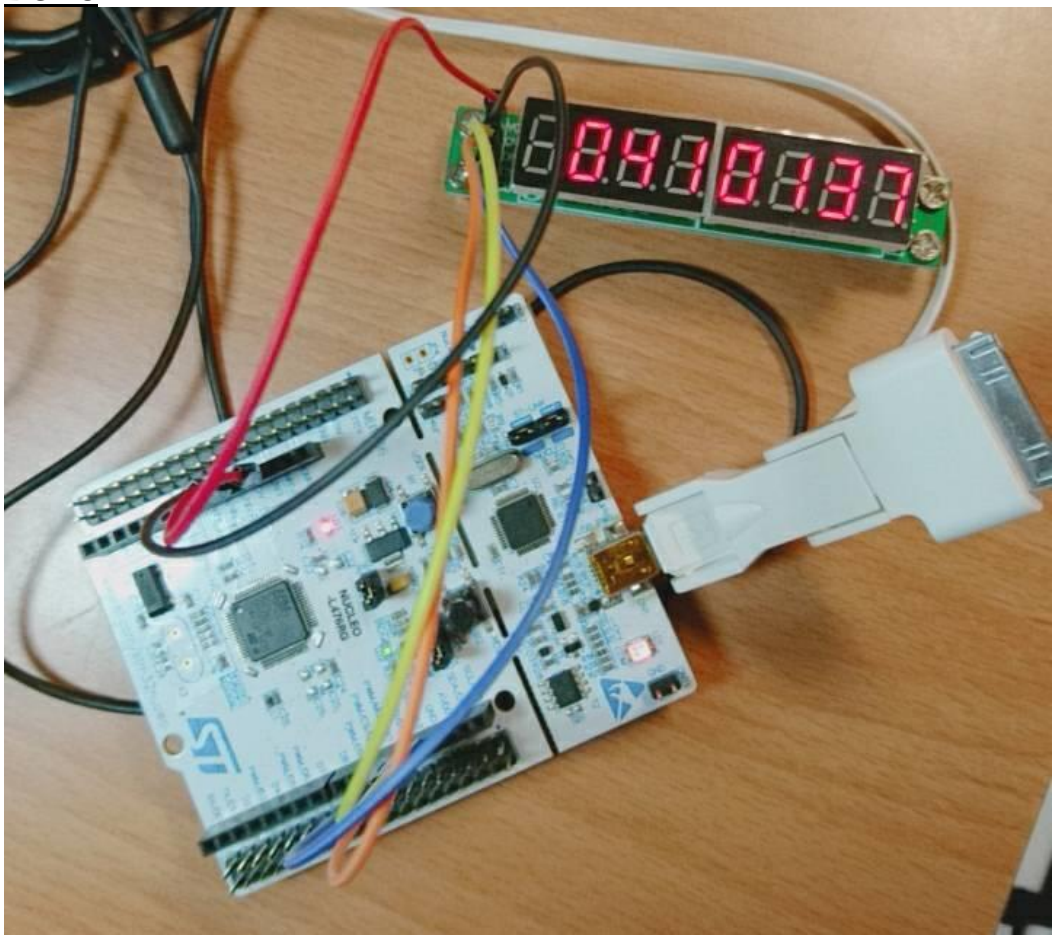
```
(just update condition flags)
   beq bit_not_set //the sending bit(r0) != 1
   str r1, [r4] //din -> 1
   b if_done

bit_not_set: //send clear bit
   str r1, [r5] //din -> 0

if_done:
   str r3, [r4] //CLK -> 1
   subs r6, 0x1
   bge send_loop
   str r2, [r5] //CS -> 0
   str r2, [r4] //CS -> 1
   pop {r0, r1, r2, r3, r4, r5, r6, r7, PC}
   BX LR
```

4. **Display_student_id**

We've used a byte array to save the student ID, so just load the array and display them on different digits.
R0 is the index of display digit.
R2 represents for student_id array index.
Digit 7 is to display the first number in student ID array whose index is 0, so we add the index of student ID array while subtracting the index of display digit.
The frequency is 100MHz, which is quite high for our eyes, and there's no delay between loops, so we can see these 7 digits at the same time.

```
Display_student_ID:
   mov r0, 0x8 //init digit = 8
   mov r2, 0x0 //arr index
   ldr r3, =student_id1
display_loop:
   subs r0, r0, 1 //digit -1
   ldrb r1, [r3,r2] //student_id1[r2]
   bl MAX7219Send
   adds r2, r2, 1 //arr index +1
   cmp r0, 1 //digit == 1
   bne display_loop
   b Display_student_ID
```

## 5. Demo



## 2.3. Lab5.3 Max7219 與 7-SEG LED 練習—顯示 Fibonacci 數

請設計一組語程式偵測實驗板上的 User button，當 User button 按 N 次時 7-Seg LED 上會顯示 fib(N)的值。User button 長按 1 秒則將數值歸零。

fib(0) = 0、fib(1) = 1、fib(2) = 1 、…

若 fib(N) ≥ 100000000 則顯示-1。

範例影片如下：

https://goo.gl/6DF6eY

Note: 請記得處理 User button 開關彈跳的問題。

Design a program to detect user button on STM32 pressed. When user button is pressed N times, display fib(N) on 7-Seg LED. When user button is held down for 1 second, set displayed number to 0. Example

video link is given above.

fib(0) = 0, fib(1) = 1, fib(2) = 1, ……

if fib(N) ≥ 100000000 then display -1.

Note: Please remember to deal with the bouncing problem.

# 1.Workflow and abstraction of this problem

step(1)

```
8       fib_ans: .asciz
  "0112358132134558914423337761098715972584418167651094617711286574636875025121393196418317811514229832040134626921783093524578570288792274651493035224157817390881696324598 6:1"
9       ans_digit: .byte 0x1, 0x1, 0x1, 0x1, 0x1, 0x1, 0x1, 0x2, 0x2, 0x2, 0x2, 0x2, 0x3, 0x3, 0x3, 0x3, 0x3, 0x4, 0x4, 0x4, 0x4, 0x5, 0x5, 0x5,
  0x5, 0x5, 0x6, 0x6, 0x6, 0x6, 0x6, 0x7, 0x7, 0x7, 0x7, 0x7, 0x8, 0x8, 0x8, 0x8, 0x2
```

build the fibonacci table (The mod is not defined in ARM, which will be quite hard to acquire each digit of fibonacci number)

(2)

Use triple pointer, ptr1 to point the current position at fib_ans for reading the data from table, ptr2 to point the current digit of the answer, which , in short is ans_digit, and finally, the ptr3 that iterates again and again in each max7219 sequence.

(3)

From 2 , e.g. the fibonacci number is now 987, than the ptr1 will point to 9

9　　8　　7

ptr1 ar 9, ptr3

ptr2=3 symbolizes the digit of current fibonacci number, for instance, the digit now is 3

and ptr 3 will now at position of 9 as well.

(4)

Once the max7219 has successfully send the data to the display, the ptr3 will increase to next position, which, to be more detailed , will get the data of next position , say 8.

Than 8 will be sent to the max7219, the same is true for all the fibonacci number in this question.

(5)

Suppose we press the use button, the ptr2 will move to the next one, to get the digit of next fibonacci number, what's more, the position of the fobonacci pointer will be moved to the position: ptr1=ptr1+ptr2

e.g. 9　8　7　1　5　9　7

ptr3------→ptr3 (move the step of digit of current fibonacci number, likewise 3)

(6)

The final position , :1 which in ascii - '0' will be turned to -1, where the fibonacci is now at #32$^{th}$ fibonacci number.

```
ldr r11, =ans_digit

push {r10}
ldr r10, =point_one_sec //trial and error
cmp r12, r10 //threshold achieved BREAKDOWN!, r6 fl
it eq
movseq r6, #1
pop {r10}

push {r10}
ldr r10, =one_sec
cmp r12, r10
it eq
movseq r6, #2
//beq clear_to_zero
pop {r10}

cmp r6, #1
it eq
ldrbeq r11, [r11,r4] //get current fibonacci digit

cmp r6, #1
it eq
addeq r4, r4, 0x1//go to next fibonacci digit

cmp r6, #1
it eq
addeq r7, r7, r11 //move to the start of next fibon

cmp r6, #2
it eq
moveq r7, #0 //move to the start of next fibonacci

cmp r6, #2
it eq
moveq r4, #0//case 2 reset all fibonacci pointers
```

(7)

In the circumstance when user button is not pressed , step 2,3,4 will be iterated again and again.

# 2.Button debounce logic will be described as follows.

(1) Once the signal is detected as 0, accumulate the counter.

(2) Once the signal reaches 1 again, reset the counter

(3) As long as the counter reach 1000, the button is confirmed as triggered, moreover, the button will be considered to be LONG PRESS as the counter reaches 10000(or some amount much higher than the normal detection of short press.

(4) In conclusion there will be two type of flag, one for short press and the other for long press.

```
check_button: //check every cycle, and accumulate 1
    ldr r5, [r8] //fetch the data from button
    lsr r5, r5, #13
    and r5, r5, 0x1 //filter the signal

    cmp r5, #0 //FUCK DONT KNOW WHY THE PRESSED SIGNAL IS 0
    it eq
    addseq r12, r12 ,#1 //accumulate until the threshol

    cmp r5, #1 //not stable, go back to accumulate again
    it eq
    moveq r12, #0
```

```
push {r10}
ldr r10, =point_one_sec
cmp r12, r10 //threshol
it eq
movseq r6, #1
pop {r10}

push {r10}
ldr r10, =one_sec
cmp r12, r10
it eq
movseq r6, #2
//beq clear_to_zero
pop {r10}
```

# 3.Demo

https://www.youtube.com/watch?v=_9CbzX6SYX0

# Acquired knowledge and thoughts

Compared to the last lab with only LED to light, this one is much more complicated ans sophisticated.  We have to fully comprehend the structure and procedure of how to place the right number on the right position in the right moment, which is extremely tough along with the debounce.

Despite the tiredness, I really learned a lot of how microprocessor works.